**Behavioral Questions**

| **Question** | **Frequency** |
| --- | --- |
| Do you use social media? What is the use of it? | 1 |
| Tell us about your hometown. | 1 |
| Why is your CGPA not very high? | 1 |
| How will you contribute to Cloud Kaptan Consultancy Services? | 1 |
| Speak about yourself non-stop | 2 |
| Tell me about yourself. | 5 |
| Family Background. | 2 |
| Why do you want to join CloudKaptan and not any other MNC? | 3 |
| Past interviews you have given and why were you not selected? | 1 |

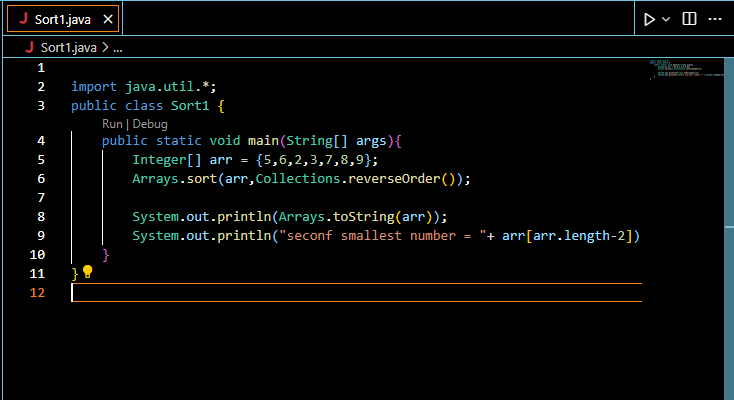
**Technical Questions (Java/OOPs/DSA)**

| **Question** | **Frequency** |
| --- | --- |
| Basics of Java Programming Language | 2 |
| Sorting of array in descending order, print 2nd smallest number | 1 |
| Difference between bubble sort and selection sort | 1 |
| Types of constructors and demonstrate using code | 1 |
| Difference between print and println in Java | 1 |
| Pillars of OOPs, explain each part | 4 |
| Search array element using recursion | 1 |
| What is abstraction? How can we implement it? | 2 |
| Reverse a string | 1 |
| Check if a matrix is an identity matrix | 1 |
| Check if two strings are anagrams | 1 |
| Code to compress string (e.g., AAABBBBBCCDDDDDD -> 3A5B2C6D) | 1 |
| How many types of inheritance are there in Java? | 1 |
| Why does Java not support multiple inheritance? | 2 |
| What is polymorphism? | 2 |
| Abstract class vs interface | 2 |
| Exception handling in Java and its hierarchy | 3 |
| Explain final, finally, finalize() | 2 |
| Code for String Palindrome | 1 |
| Static vs constant | 2 |
| JDK-JRE-JVM Comparison | 1 |
| Code to find the nth node in a linked list | 1 |
| Code to rotate a matrix by 180° and print the diagonal sum | 1 |
| Sort array in ascending order up to an index, then descending | 1 |
| Magic Number problem | 1 |
| GCD calculation | 1 |
| Print pattern | 1 |
| Write the fibbonacci series code | 1 |
| Compare C vs Java | 1 |
| Tic Tac Toe by JAVA | 1 |
| Interface Usage  Custom exception  Integer to roman and vice versa  Multithreading and Lifecycle of Threads |  |

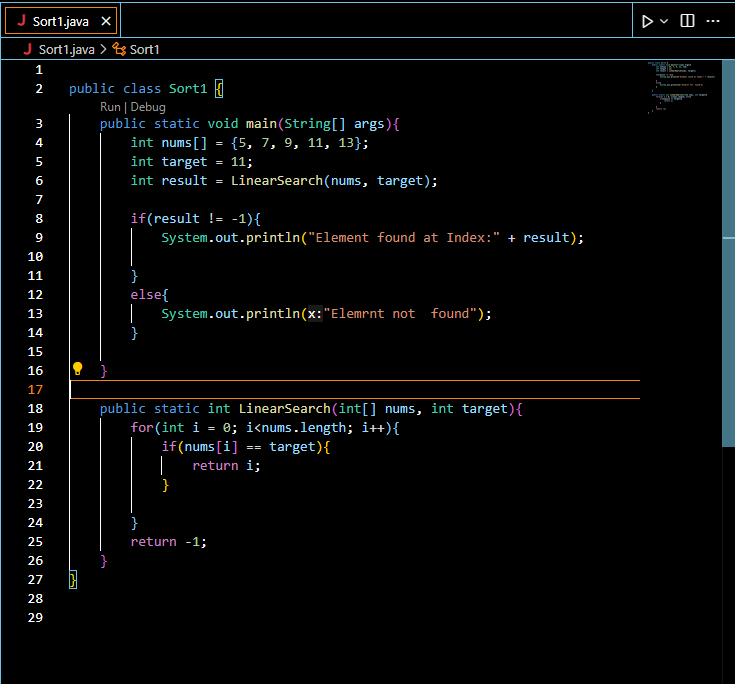
* Sorting in Ascending.



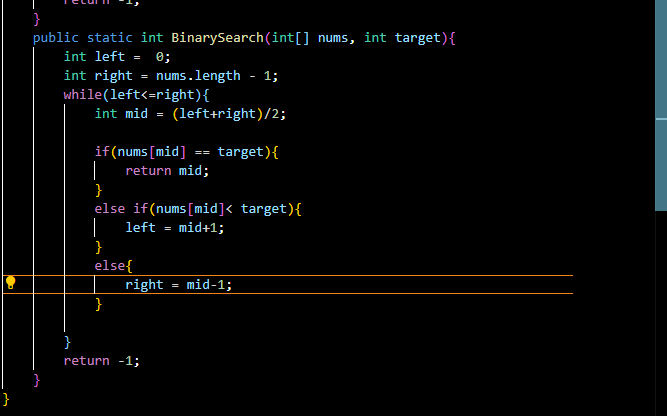
* Sorting in descending

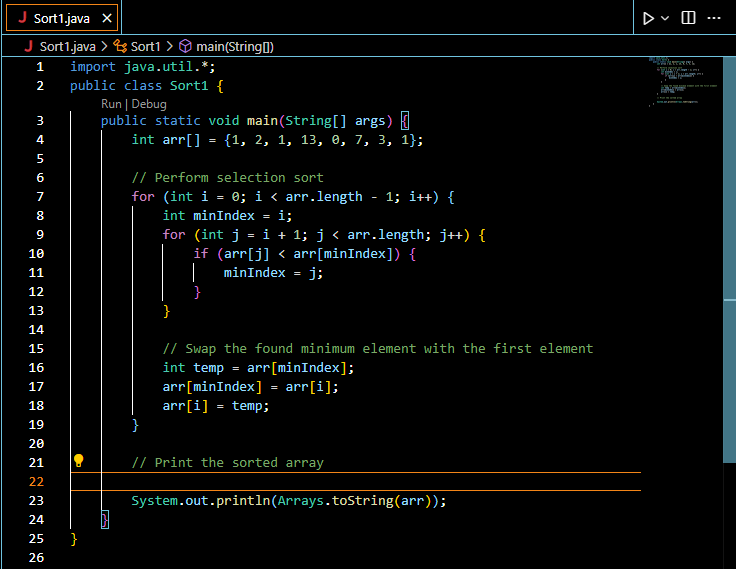


Linear Search:



Binary Search



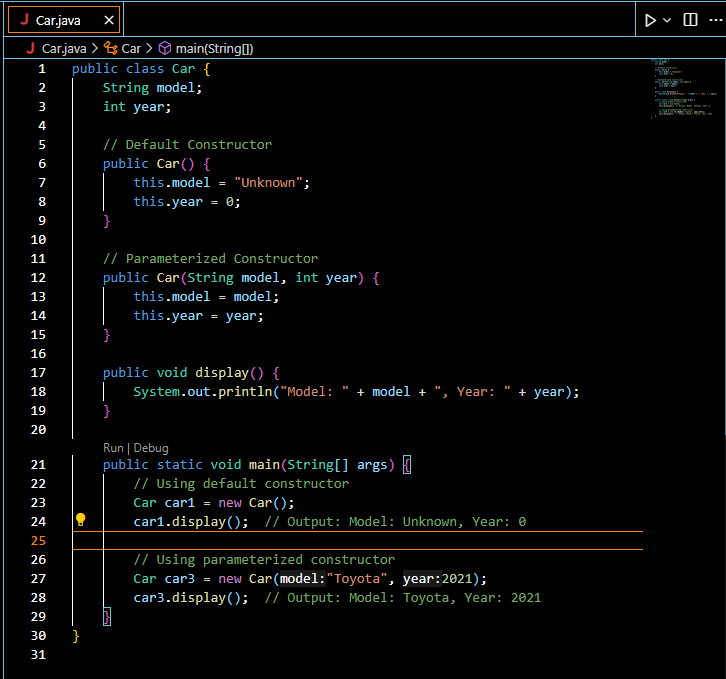
Selection sort:  


Types of Constructors:

 **Default Constructor**: A no-argument constructor automatically provided by Java if no other constructor is defined.

 **No-Argument Constructor**: A user-defined constructor with no parameters.

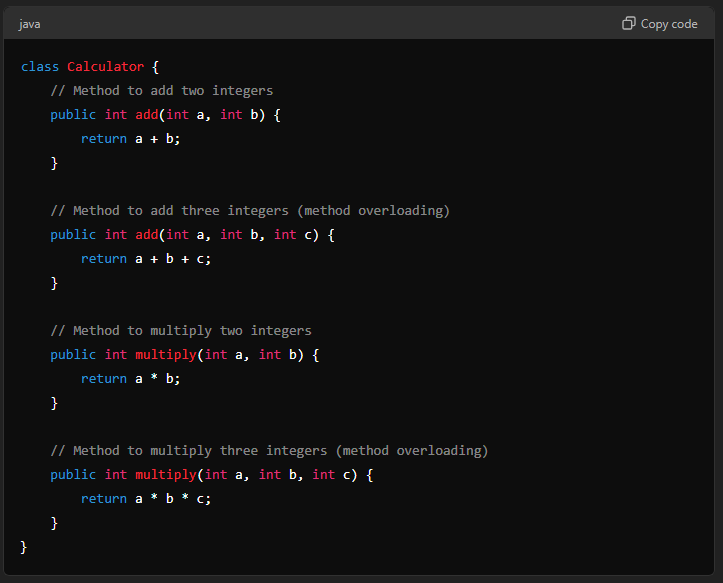
 **Parameterized Constructor**: A constructor with parameters, allowing the initialization of objects with specific values.

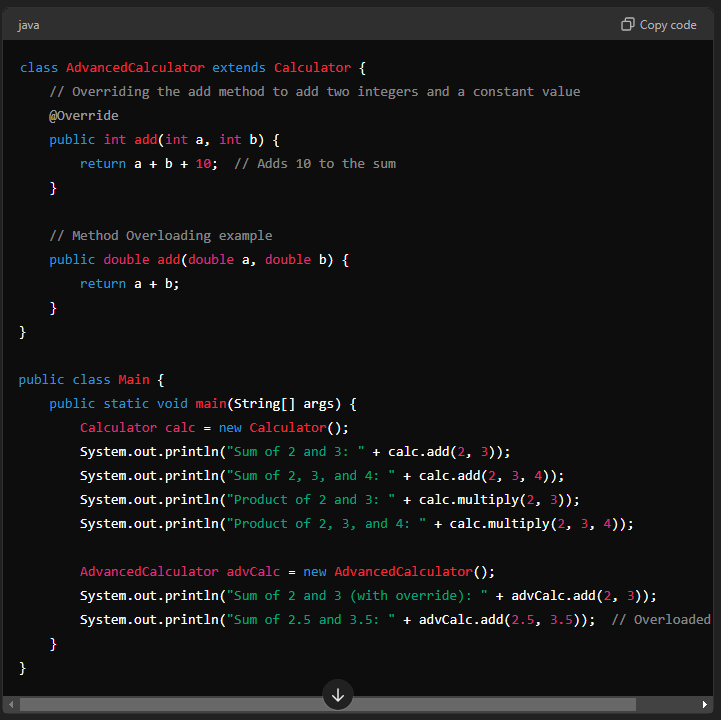


The four pillars of Object-Oriented Programming (OOP) in Java are:

1. **Encapsulation**: Encapsulation involves bundling the data (variables) and methods that operate on the data into a single unit or class. It also restricts access to some of the object's components to protect the object's integrity. Example: Using private fields and public getter/setter methods.
2. **Inheritance**: Inheritance allows a new class (subclass) to inherit attributes and methods from an existing class (superclass), promoting code reusability. Example: class Dog extends Animal inherits properties from Animal.
3. **Polymorphism**: Polymorphism allows objects to be treated as instances of their parent class, with the ability to override methods to provide specific behavior. Example: Method overriding and overloading.
4. **Abstraction**: Abstraction hides complex implementation details and shows only essential features. This can be achieved through abstract classes or interfaces. Example: Defining an interface Animal with abstract methods that different animal classes implement.

These pillars work together to create flexible, maintainable, and reusable code in Java.





Method overloading vs overriding;

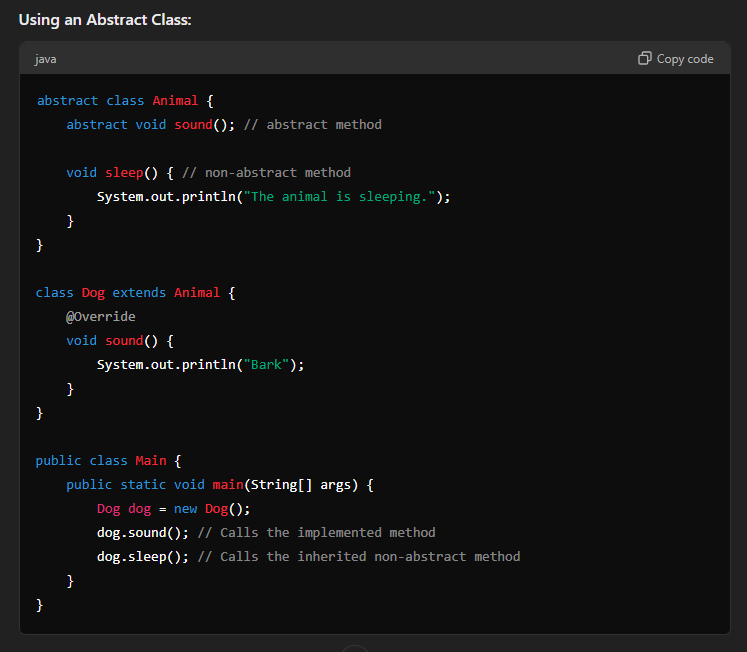
Recursion search in an array;

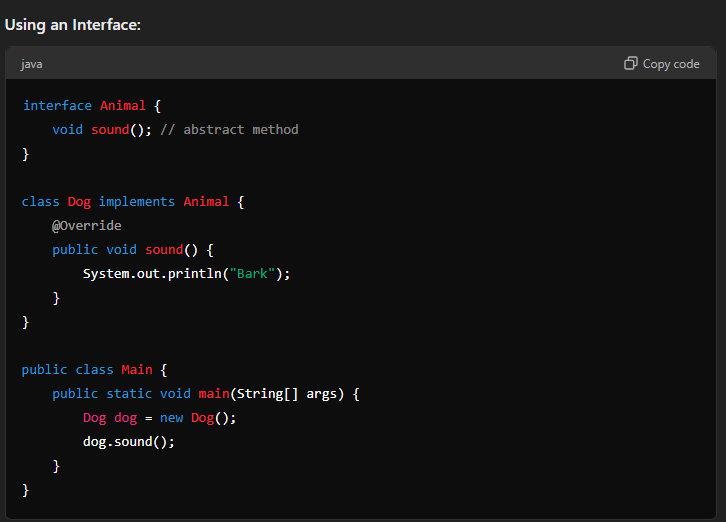


**Abstraction** in Java is a fundamental concept in object-oriented programming (OOP). It involves hiding the complex implementation details of a class and exposing only the essential features to the user. This allows you to focus on what an object does rather than how it does it.

**Implementation of Abstraction:**

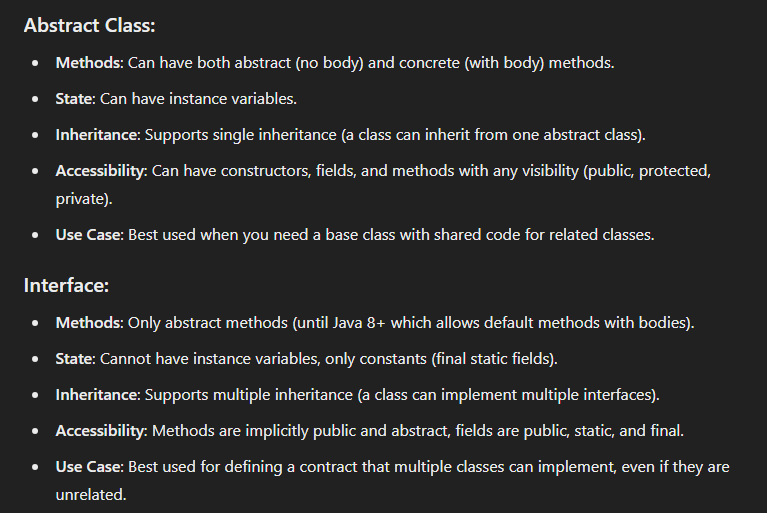
1. **Abstract Class**: A class that cannot be instantiated on its own and may contain abstract methods (methods without implementation) that must be implemented by its subclasses.
2. **Interface**: A completely abstract class that defines methods that must be implemented by any class that implements the interface.

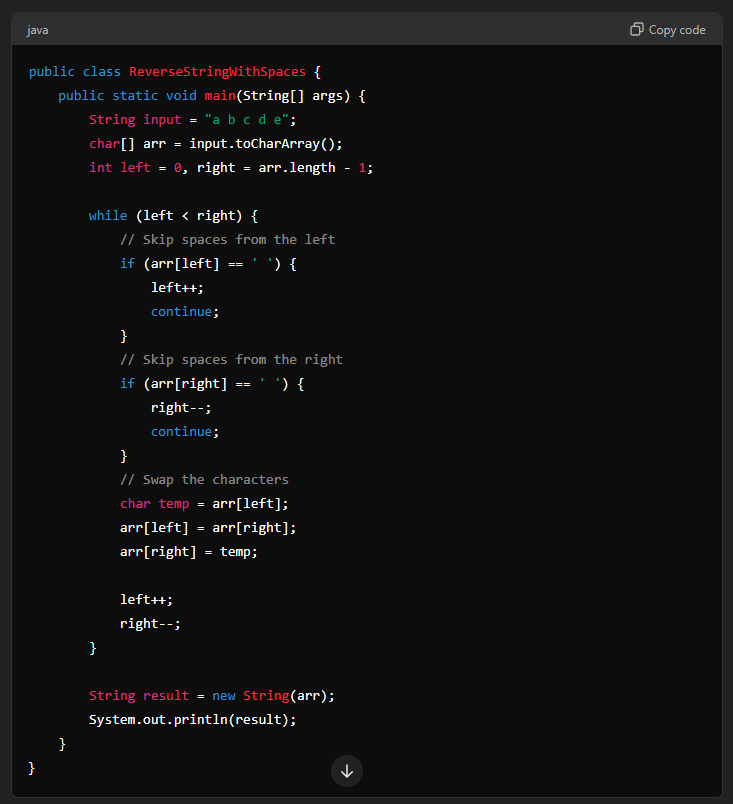


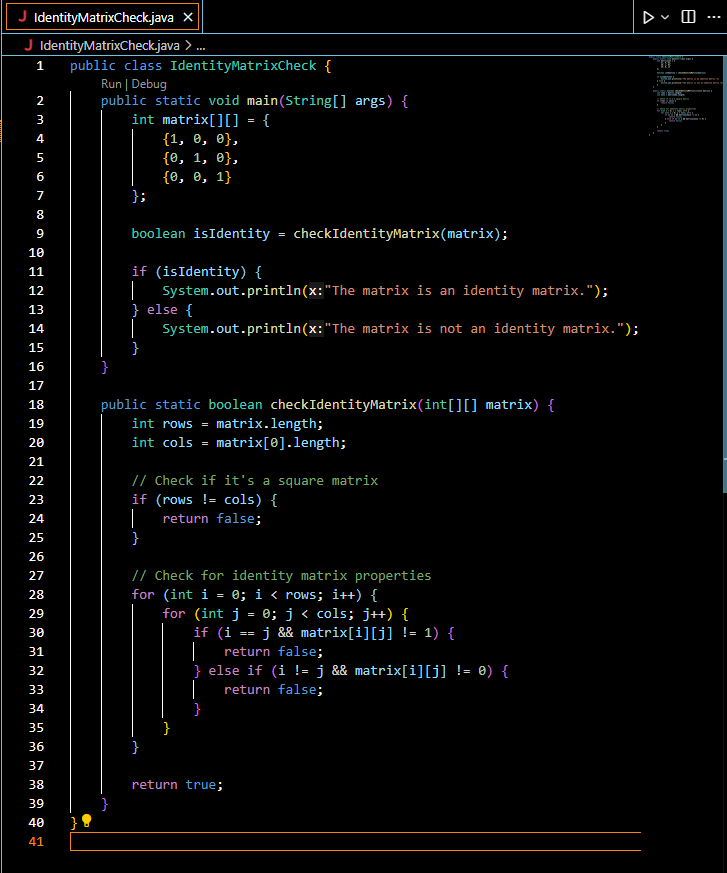


**Key Points:**

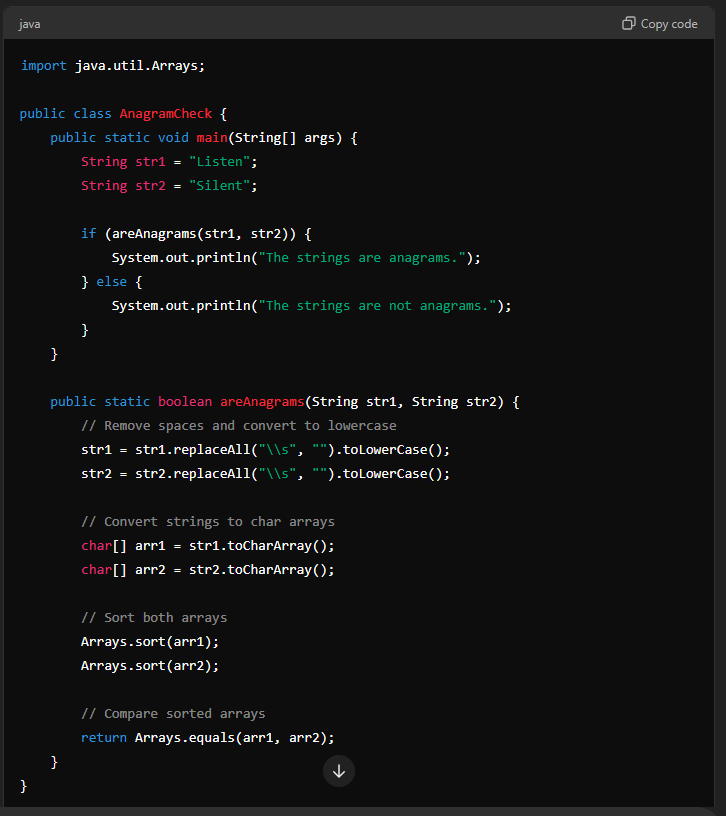
* **Abstract classes** can have both abstract and non-abstract methods, while **interfaces** can only have abstract methods (though Java 8+ allows default methods with implementation in interfaces).
* Abstract classes are used when there is a common base class, but you don't want to allow creating objects of that base class.
* Interfaces are used to implement multiple inheritance and define a contract that the implementing classes must follow.

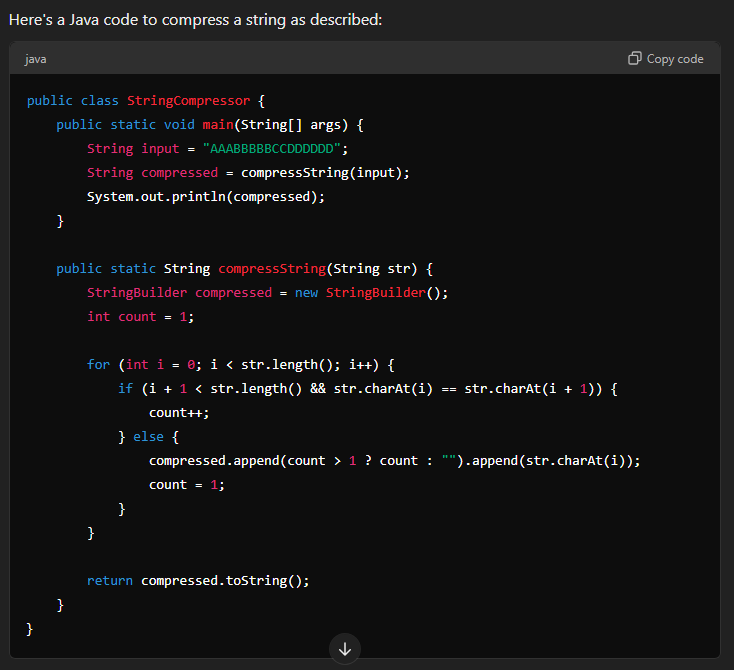


Reverse a string keeping the Spaces at place  


Identity Matrix or not?  


String anagrams or not?:  
nagrams are words or phrases formed by rearranging the letters of another word or phrase, typically using all the original letters exactly once. For example, the word "listen" can be rearranged to form the word "silent." Both words use the same letters, but in a different order, making them anagrams of each other.

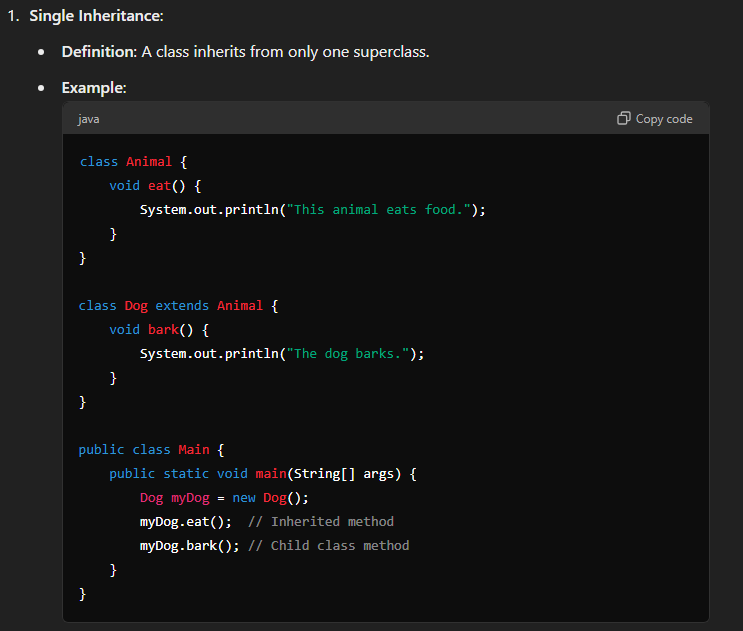
  
  
Compress string

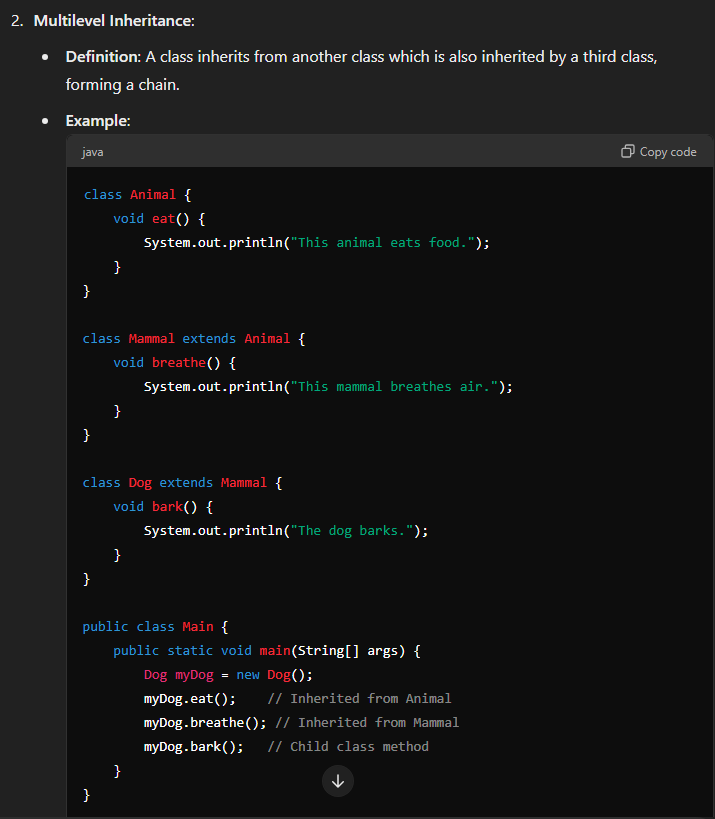


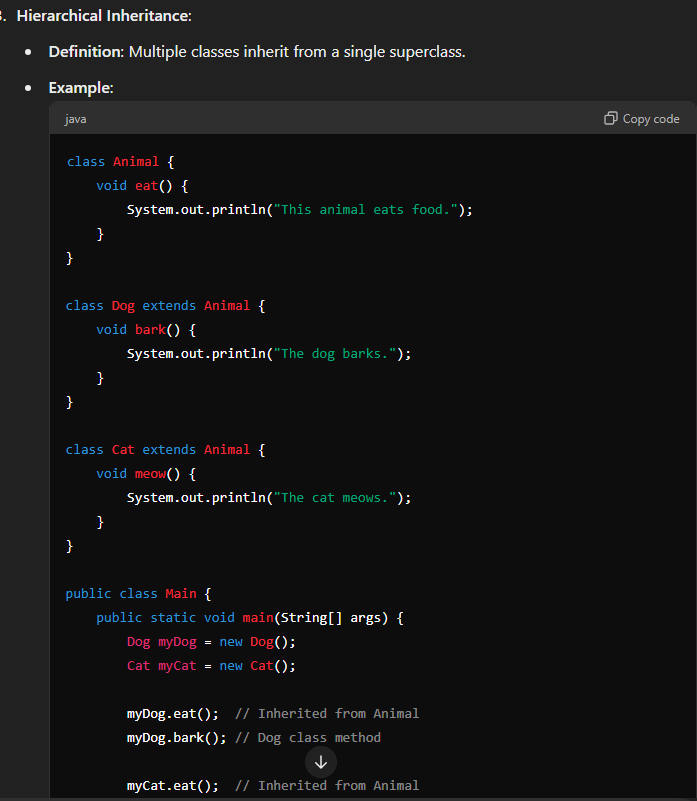
**Why Java Does Not Support Multiple Inheritance**

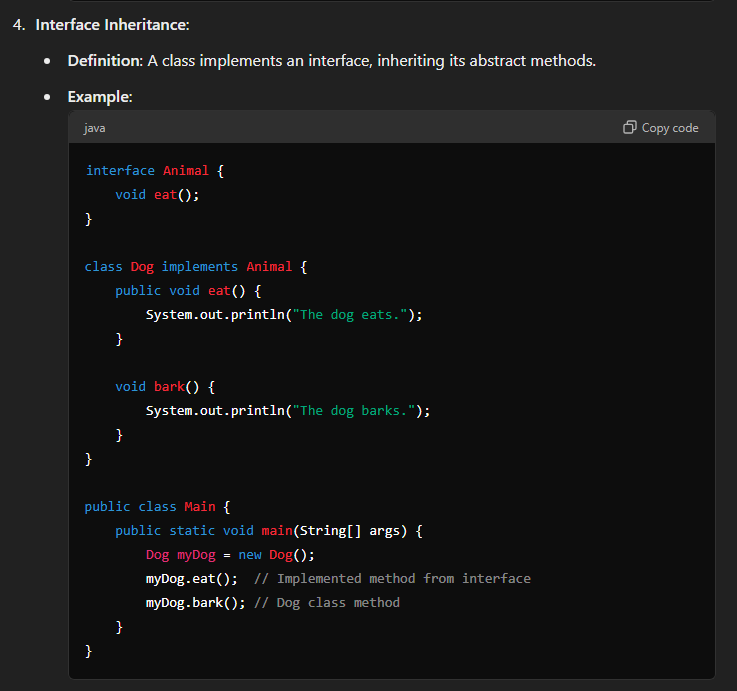
Java does not support multiple inheritance (where a class inherits from more than one class) due to several reasons:

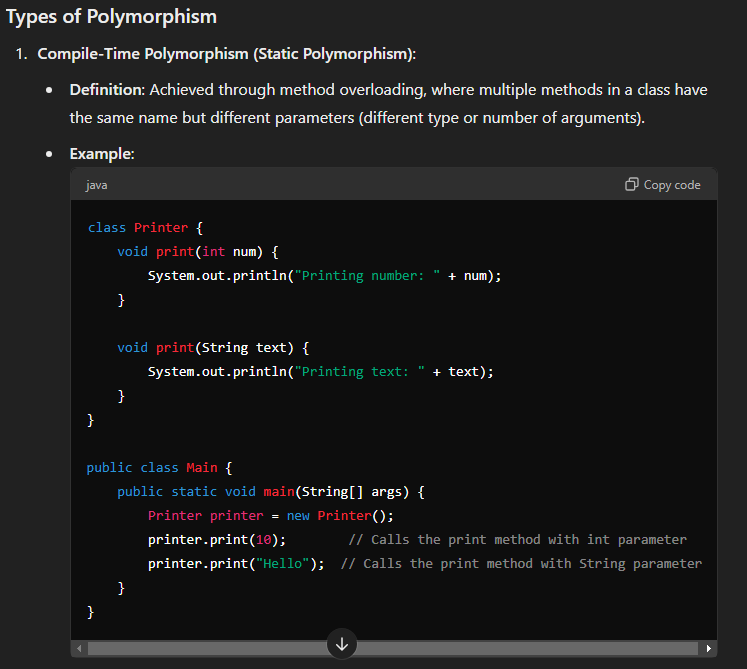
1. **Diamond Problem**:
   * When a class inherits from multiple classes, there is ambiguity about which method implementation should be used if the parent classes have methods with the same signature. This problem is known as the "Diamond Problem."
2. **Simplicity and Design**:
   * Avoiding multiple inheritance simplifies the language design and reduces complexity in understanding and maintaining the code.
3. **Interface Solution**:
   * Java allows multiple inheritance of types through interfaces. A class can implement multiple interfaces, which provides the flexibility of multiple inheritance without the associated complexity. Interfaces only declare methods but do not provide implementations, thereby avoiding the diamond problem.









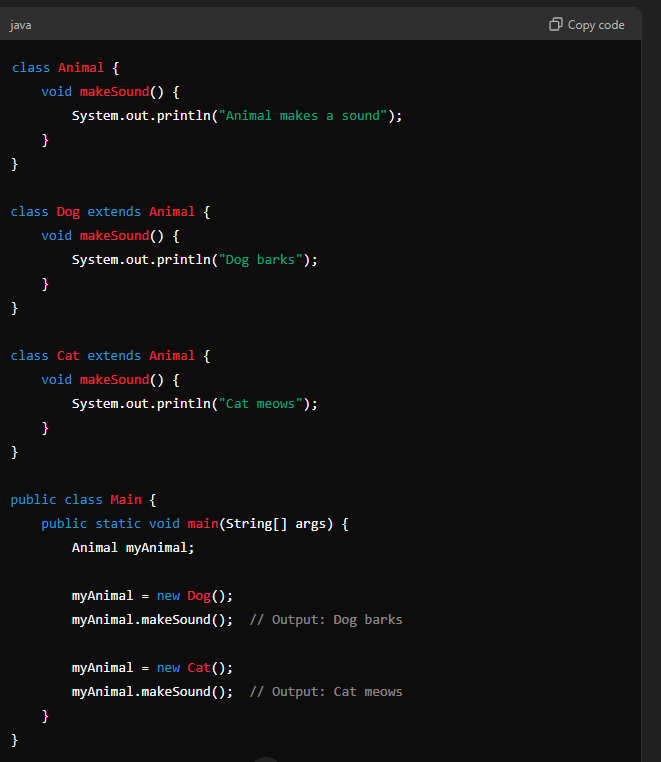


**Run-Time Polymorphism (Dynamic Polymorphism)**:

* **Definition**: Achieved through method overriding, where a subclass provides a specific implementation of a method that is already defined in its superclass. The method that gets executed is determined at runtime based on the object’s actual type.

**Key Points of Polymorphism**

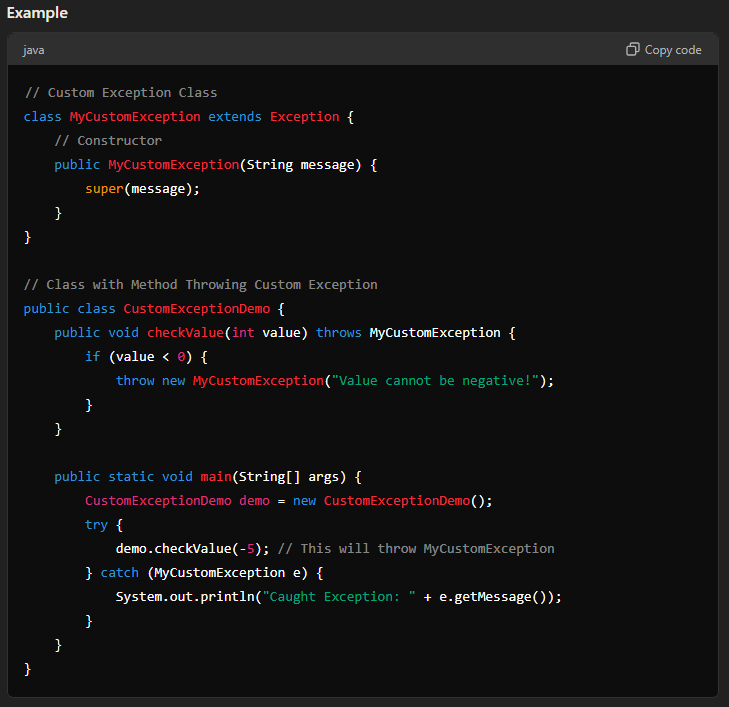
* **Flexibility**: Polymorphism allows you to write more flexible and reusable code. You can use the same method name to perform different tasks based on the object type or method parameters.
* **Method Overloading vs. Overriding**: Overloading is a compile-time concept, while overriding is a runtime concept. Overloading allows methods with the same name but different parameters in the same class, whereas overriding allows a subclass to provide a specific implementation of a method defined in its superclass.
* **Dynamic Method Dispatch**: In dynamic polymorphism, the method that gets called is determined at runtime based on the object type, not the reference type.

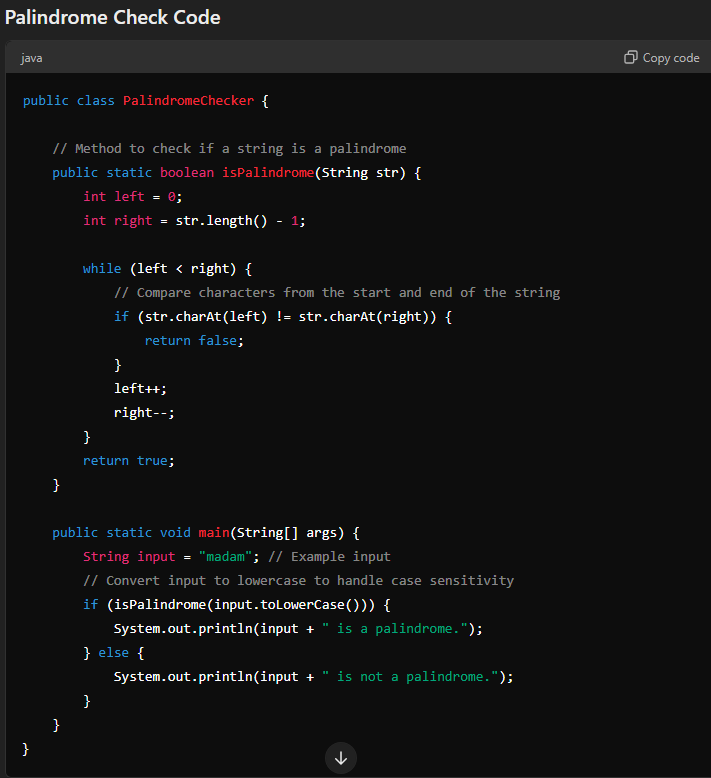


Exception handling:



Custom Exception:



Explain final, finally, finalize()  
chat gpt:  


Custom exeption:



Final finally and finalize();  
  